## Подключение к базе данных

В любом проекте WPF, как и в ряде других типов проектов для .NET, по умолчанию есть файл конфигурации, который называется *app.config* и который имеет следующее содержимое:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <?xml version="1.0" encoding="utf-8" ?>  <configuration>      <startup>          <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />      </startup>  </configuration> |

Добавим в него строку подключения к бд, изменив файл следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?xml version="1.0" encoding="utf-8" ?>  <configuration>      <startup>          <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />      </startup>    <connectionStrings>     <add name="DefaultConnection"          connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=mobiledb;Integrated Security=True"         providerName="System.Data.SqlClient"/>    </connectionStrings>  </configuration> |

Для определения всех подключений в программе в пределах узла <configuration> добавляется новый узел <connectionStrings>. В этом узле определяются строки подключения с помощью элемента <add>. Каждая строка подключения имеет название, определяемое с помощью атрибута name. В данном случае строка подключения называется "DefaultConnection". Название может быть произвольное.

Атрибут connectionString собственно хранит строку подключения. Он состоит из трех частей:

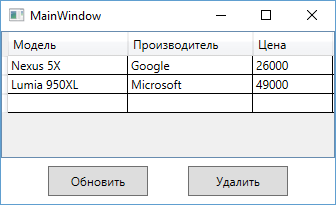
* Data Source=.\SQLEXPRESS: указывает на название сервера. По умолчанию для MS SQL Server Express используется ".\SQLEXPRESS"
* Initial Catalog=mobiledb: название базы данных. Так как база данных называется mobiledb, то соответственно здесь данное название и указываем
* Integrated Security=True: задает режим аутентификации

Так как мы будем подключаться к базе данных MS SQL Server, то соответственно мы будем использовать провайдер для SQL Server, функциональность которого заключена в пространстве имен System.Data.SqlClient.

Далее определим код графического интерфейса в xaml:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | <Window x:Class="DbApp.MainWindow"          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"          xmlns:d="http://schemas.microsoft.com/expression/blend/2008"          xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"          xmlns:local="clr-namespace:DbApp"          mc:Ignorable="d"          Title="MainWindow" Height="250" Width="350" Loaded="Window\_Loaded">      <Window.Resources>          <Style TargetType="Button">              <Setter Property="Margin" Value="20 8 20 8" />              <Setter Property="Width" Value="100" />              <Setter Property="Height" Value="30" />          </Style>      </Window.Resources>      <Grid>          <Grid.RowDefinitions>              <RowDefinition Height="\*" />              <RowDefinition Height="Auto" />          </Grid.RowDefinitions>          <DataGrid AutoGenerateColumns="False" x:Name="phonesGrid">              <DataGrid.Columns>                  <DataGridTextColumn Binding="{Binding Title}" Header="Модель" Width="120"/>                  <DataGridTextColumn Binding="{Binding Company}" Header="Производитель" Width="125"/>                  <DataGridTextColumn Binding="{Binding Price}" Header="Цена" Width="80"/>              </DataGrid.Columns>          </DataGrid>            <StackPanel HorizontalAlignment="Center"  Grid.Row="1" Orientation="Horizontal">              <Button x:Name="updateButton" Content="Обновить" Click="updateButton\_Click" />              <Button x:Name="deleteButton" Content="Удалить" Click="deleteButton\_Click" />          </StackPanel>      </Grid>  </Window> |

Здесь определен довольно простой интерфейс: датагрид для отображения данных, и две кнопки для обновления данных в бд и для удаления. В итоге приложение будет выглядеть следующим образом:



Теперь определим код подключения и все обработчики кнопок в файле кода c#:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86 | using System;  using System.Windows;  using System.Windows.Controls;  using System.Data.SqlClient;  using System.Data;  using System.Configuration;    namespace DbApp  {      public partial class MainWindow : Window      {          string connectionString;          SqlDataAdapter adapter;          DataTable phonesTable;            public MainWindow()          {              InitializeComponent();              // получаем строку подключения из app.config              connectionString = ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;          }            private void Window\_Loaded(object sender, RoutedEventArgs e)          {              string sql = "SELECT \* FROM Phones";              phonesTable = new DataTable();              SqlConnection connection=null;              try              {                  connection = new SqlConnection(connectionString);                  SqlCommand command = new SqlCommand(sql, connection);                  adapter = new SqlDataAdapter(command);                    // установка команды на добавление для вызова хранимой процедуры                  adapter.InsertCommand = new SqlCommand("sp\_InsertPhone", connection);                  adapter.InsertCommand.CommandType = CommandType.StoredProcedure;                  adapter.InsertCommand.Parameters.Add(new SqlParameter("@title", SqlDbType.NVarChar, 50, "Title"));                  adapter.InsertCommand.Parameters.Add(new SqlParameter("@company", SqlDbType.NVarChar, 50, "Company"));                  adapter.InsertCommand.Parameters.Add(new SqlParameter("@price", SqlDbType.Int, 0, "Price"));                  SqlParameter parameter = adapter.InsertCommand.Parameters.Add("@Id", SqlDbType.Int, 0, "Id");                  parameter.Direction = ParameterDirection.Output;                    connection.Open();                  adapter.Fill(phonesTable);                  phonesGrid.ItemsSource = phonesTable.DefaultView;              }              catch(Exception ex)              {                  MessageBox.Show(ex.Message);              }              finally              {                  if(connection!=null)                      connection.Close();              }          }            private void UpdateDB()          {              SqlCommandBuilder comandbuilder = new SqlCommandBuilder(adapter);              adapter.Update(phonesTable);          }            private void updateButton\_Click(object sender, RoutedEventArgs e)          {              UpdateDB();          }            private void deleteButton\_Click(object sender, RoutedEventArgs e)          {              if (phonesGrid.SelectedItems != null)              {                  for (int i = 0; i < phonesGrid.SelectedItems.Count; i++)                  {                      DataRowView datarowView = phonesGrid.SelectedItems[i] as DataRowView;                      if (datarowView != null)                      {                          DataRow dataRow = (DataRow)datarowView.Row;                          dataRow.Delete();                      }                  }              }              UpdateDB();          }      }  } |

Вся работа с бд производится стандартными средствами ADO.NET и прежде всего классом SqlDataAdapter. Вначале мы получаем в конструкторе строку подключения, которая определена выше в файле *app.config*:

|  |  |
| --- | --- |
| 1 | connectionString = ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString; |

Чтобы задействовать эту функциональность, нам надо добавить в проект библиотеку **System.Configuration.dll**.

Далее в обработчике загрузки окна Window\_Loaded создаем объект SqlDataAdapter:

|  |  |
| --- | --- |
| 1 | adapter = new SqlDataAdapter(command); |

В качестве команды для добавления объекта устанавливаем ссылку на хранимую процедуру:

|  |  |
| --- | --- |
| 1 | adapter.InsertCommand = new SqlCommand("sp\_InsertPhone", connection); |

Получаем данные из БД и осуществляем привязку:

|  |  |
| --- | --- |
| 1  2 | adapter.Fill(phonesTable);  phonesGrid.ItemsSource = phonesTable.DefaultView; |

За обновление отвечает метод UpdateDB():

|  |  |
| --- | --- |
| 1  2  3  4  5 | private void UpdateDB()  {      SqlCommandBuilder comandbuilder = new SqlCommandBuilder(adapter);      adapter.Update(phonesTable);  } |

Чтобы обновить данные через SqlDataAdapter, нам нужна команда обновления, которую можно получить с помощью объекта SqlCommandBuilder. Для самого обновления вызывается метод adapter.Update().

Причем не важно, что мы делаем в программе - добавляем, редактируем или удаляем строки. Метод adapter.Update сделает все необходимые действия. Дело в том, что при загрузке данных в объект DataTable система отслеживает состояние загруженных строк. В методе adapter.Update() состояние строк используется для генерации нужных выражений языка SQL, чтобы выполнить обновление базы данных.

В обработчике кнопки обновления просто вызывается этот метод UpdateDB, а в обработчике кнопки удаления предварительно удаляются все выделенные строки.

Таким образом, мы можем вводить в DataGrid новые данные, редактировать там же уже существующие, сделать множество изменений, и после этого нажать на кнопку обновления, и все эти изменения синхронизируются с базой данных.

Причем важно отметить действие хранимой процедуры - при добавлении нового объекта данные уходят на сервер, и процедура возвращает нам id добавленной записи. Этот id играет большую роль при генерации нужного sql-выражения, если мы захотим эту запись изменить или удалить. И если бы не хранимая процедура, то нам пришлось бы после добавления данных загружать заново всю таблицу в datagrid, только чтобы у новой добавленной записи был в datagrid id. И хранимая процедура избавляет нас от этой работы.

Также здесь мы могли бы выполнять обновление данных сразу после редактирования строки. Для этого нужно задействовать событие RowEditEnding элемента DataGrid:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public MainWindow()  {      InitializeComponent();        connectionString = ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;      phonesGrid.RowEditEnding += PhonesGrid\_RowEditEnding;  }    private void PhonesGrid\_RowEditEnding(object sender, DataGridRowEditEndingEventArgs e)  {      UpdateDB();  } |

И если после окончания редактирования мы нажмем на Enter, то срабатает обработчик события RowEditEnding, который обновит базу данных.

Итак, здесь рассмотрен простейший способ работы с базой данных в WPF. Далее мы рассмотрим еще один способ, который подразумевает применение технологии Entity Framework.